data preprocess: from raw file(training data tsv + mention-candidate tsv) to json file

see preprocess.py

create_mention_candid_dict: get dict{mention:[candidate set]}(original candidates, not lower ones)
get_local_cnt: split document context in raw file(tsv), get 10 words + [mention] + 10 words

when getting candidates: if this mention is not in mention_candid_dict, add its gold answer(lower) to its candidate list and set gold_idx to 0

when adding candidates to the value of 'candidates' of a key, the candidate name will be lowercases. Find the position of gold answer and assign it to 'gold_idx'

create_candid_doc_dict: after collecting all candidates of the dataset, extract candidate-document context from wiki dump, build a dictionary {candidate_name(lower) : wiki_document context} and save it to a pickle file

save the dictionary to a json file and then split the file to sub files(train, dev, test)

word embedding

see runner.py

if the embedding file is .txt, we just read each line, save the location of words to a word_location_dictionary{word: location index} and save the word embedding matrix to a np.array.
if the embedding file is binary file, we will use method that can read binary file, the operations are different but the procedure is the same.
I add an <emp> word at the end of the matrix and its word embedding is [0] * word embedding dimension.

data preprocess: from json file to batchobject that can directly feed to the CNN model

see linker_train.py jsonTobatch

each mention, its local context and its document context will copy its number of candidates times. After getting mention, local context, document context, candidate name and candidates wiki document context, they will be converted to IDs according to the word embedding matrix, if the word doesn't exist in word_location_dictionary, map it to <emp>. They will have specific length. Note that if a candidate cannot find its document, its document context will be all zero.
also, y_isgold indicates which candidate is the gold answer and y_grouping stores the start index, end index and the gold index of this mention
Save all these IDs to a batchobject.

the function will return a list which contains all batchobjects.

run process

see linker_train.py

update 'candidate_scores' after each iteration.

evaluation

see evaluation.py

evaluate the result after 'candidate_scores' of all batches have been updated.

CNN model

see my_cnn.py